

# MEMORY EFFICIENT PASS-PARALLEL ARCHITECTURE FOR JPEG2000 ENCODING

*Michael Dyer, David Taubman, Saeid Nooshabadi*

School of Electrical Engineering  
University of New South Wales  
Sydney, Australia

## ABSTRACT

The CAUSAL, RESTART and RESET mode switches, previously used to enable microscopic parallelism and improve throughput, are examined in terms of the memory requirements of the JPEG2000 block coder. An Extended Pass Switching Arithmetic Encoder (EPSAE) is introduced that aids in the reduction of memory by providing the ability to partially process code-blocks. We show how the use of these switches and the EPSAE can reduce the overall amount of memory required by the block coder by a factor of 7. This reduction is achieved without the necessity of tight synchronization between the DWT and block coder.

## 1. INTRODUCTION

This paper demonstrates the use of the RESTART, RESET and CAUSAL mode switches (the stripe causal method [1]) in the reduction of memory (in particular, subband data memory) required by a JPEG2000 encoder. Previously [1] [2], these switches have been described as a means of providing parallelism and improving throughput.

We analyze three systems. In the first, the Discrete Wavelet Transform (DWT) engine is loosely coupled to the block coding sub-system. The second uses a tight coupling between these two sub-systems. The third lies between these two extremes, utilizing an Extended Pass Switching Arithmetic Encoder (EPSAE) to reduce the amount of buffering required for subband samples. A benefit of this technique is that the DWT can remain loosely synchronized with the block coder.

The EPSAE is an extension to the Pass Switching Arithmetic Encoder (PSAE) proposed in [2]. Unlike the PSAE, the proposed coder enables partial coding of code-blocks. It is this property that enables the proposed system to use approximately 7 times less memory than the loosely coupled system.

## 2. THE DWT ENGINE

Before comparing the three schemes, it is important to outline the parameters affecting the memory requirements of

the DWT engine.

We assume that the DWT engine is capable of producing  $M$  lines of subband samples at once, from  $(2L_{max} + 2M - 1)$  lines of input samples, where  $(2L_{max} + 1)$  is the length of the longest wavelet filter.

The following equations (summarized from [3]) denote the cost of buffering the horizontal and vertical low-pass intermediate subband data. For simplicity, they assume an original image bit depth of 8 bits, and a subband sample bit depth of 16 bits. The original image width is  $W$  samples.

The buffer size for implementing a direct DWT analysis engine is given as

$$S_{DWT}^D \lesssim (6M + 6L_{max} - 3)W \text{ bytes}$$

For a lifting based engine, the buffer size is

$$S_{DWT}^L \lesssim (6M + 4L_{max} - 1)W \text{ bytes}$$

Memory bandwidth (the number of memory transactions required per original image sample) for the direct implementation is

$$BW_{DWT}^D \lesssim \frac{5}{3} \left( 2 + \frac{2L_{max} - 1}{2M} \right) \text{ bytes/sample}$$

For the lifting based engine, bandwidth is

$$BW_{DWT}^L \lesssim \frac{5}{3} \left( 2 + \frac{2L_{max} - 1}{2M} \right) + \frac{L_{max} - 1}{M} \text{ bytes/sample}$$

It is important to note that the size of external buffers increases by  $6W$  bytes for each extra line. For example, a 1024 sample wide image, with  $M = 4$  and  $L_{max} = 4$  will require 45kB (39kB) of memory for the direct (lifting) implementation. For  $M = 64$ , 405kB(399kB) are required. We also note that the lifting implementation has a slightly higher external memory bandwidth than the direct implementation.

## 3. FORMS OF COUPLING

In this paper, coupling refers to the way the DWT is connected to a pool of block coders. The form of coupling

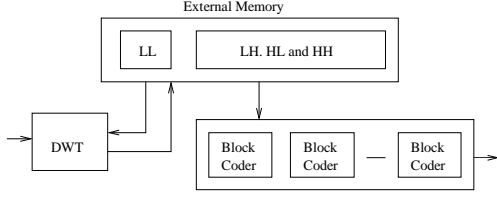


Figure 1: Loose Coupling

influences the amount of memory and memory bandwidth required by the system.

The loosely coupled system of Figure 1 assumes the DWT produces a small number of lines ( $M < J$ , where  $J$  is the height of the code-block) of subband samples at once. These are stored in a subband data memory, from which they are consumed by the pool of block coders. The subband data memory must be able to accommodate at least  $J$  lines from each subband, at each level of the DWT, so that the block coders can process complete code-blocks. This system allows the DWT memory cost to be kept low, at the expense of a large subband data memory.

In a tightly coupled system (Figure 2), the DWT provides subband data directly to the pool of block coders. This suggests that the DWT should produce  $J$  lines at once for each subband in the DWT level which is currently being processed. This system eliminates the need for any significant subband data memory, at the expense of a much higher memory cost for the DWT. While the techniques described in this paper could be applied to reduce the memory cost associated with the tightly coupled system, tight coupling has a significant disadvantage. Specifically, enough block coders must be provided to ensure that the required throughput can be maintained. Unfortunately, the block coding time can vary greatly from block to block, based on the statistics of the subband. As a result, many block coders may be idle much of the time.

The system proposed in this paper is shown in Figure 3. The block coder is more tightly coupled to the DWT than in the loose system, but simple synchronization is maintained, through the use of double buffering. This system exploits the fact that code-block samples are processed in stripes, having height 4. Sufficient buffering is provided for one or more stripes of subband data at the highest decomposition level. The DWT produces  $M = 4S$  lines at once, where  $S$  is the number of stripes that are processed together by the block coder. For this system to work, the CAUSAL, RESTART and RESET mode switches must be employed, breaking key dependencies in the block coder.

#### 4. SUBBAND DATA MEMORY

For the loosely coupled system, enough memory must be provided to at least buffer a row of code-blocks from each subband at each level of decomposition. Assuming 16 bit

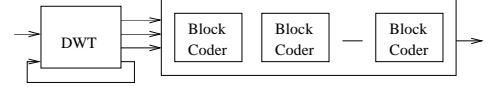


Figure 2: Tight Coupling

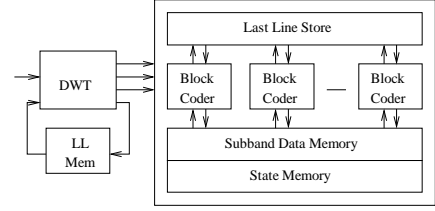


Figure 3: Proposed Coupling

subband samples,  $D$  levels of decomposition and double buffering, the amount of subband data memory required is

$$S_{\text{Subband}}^{\text{Loose}} = 12JW \sum_{d=1}^D 2^{-d} \lesssim 12JW \text{ bytes}$$

For example, a 1024 wide image, using  $64 \times 64$  sample code-blocks and using 5 levels of decomposition requires 744kB of subband data memory.

Each subband sample must be read and written once. There are the same number of subband samples as image samples. Assuming 16 bit subband samples, the required bandwidth for the subband data memory is

$$BW_{\text{Subband}}^{\text{Loose}} = 4 \text{ bytes/sample}$$

The proposed system requires enough memory to buffer a group of stripes at any single level of decomposition. Again, assuming double buffering and 16 bit subband samples, we obtain

$$S_{\text{Subband}}^{\text{Proposed}} = 24SW \text{ bytes}$$

For example, a 1024 wide image, with the coders processing using a single stripe ( $S = 1$ ), will require 24kB of subband data memory.

As for the loose system, the bandwidth is

$$BW_{\text{Subband}}^{\text{Proposed}} = 4 \text{ bytes/sample}$$

although the impact of this memory bandwidth may be much less significant than in the loosely coupled system, since the subband data memory may be small enough to reside on-chip.

#### 5. EXTENDED PASS SWITCHING ARITHMETIC ENCODER

The proposed system of Figure 3 must be able to process partial code-blocks. Specifically, the block coder must be able to leave a code-block when it runs out of stripe data and return to it later when more data is available. This means that the system must be able to store and retrieve

state information corresponding to each coding pass of each code-block which has been partially processed.

The Pass Switching Arithmetic Encoder (PSAE) was originally introduced in [2] as an alternative to the coding primitives outlined in [4] for reducing wasted clock cycles. Extending this idea to allow the MQ registers and probability models (the coder “state”) to be stored between passes enables the block coder to operate on partial code-blocks. Each pass state is stored in “State Memory” shown in Figure 3. By enabling the RESTART and RESET mode switches, the MQ codeword is terminated and the probability models are reset at the end of each pass, making the arithmetic coding processes independent across passes. Additionally, because we are operating on partial code-blocks, the CAUSAL mode switch is needed to remove the possibility that future stripes contribute to the generation of context symbols for the current stripe.

Using the notation of [3], the MQ registers are  $A$ ,  $C$ ,  $\bar{t}$ ,  $\bar{T}$  and  $L$ . As will be shown in Section 6, it is not necessary to store  $L$  in state memory. The others require 51 bits per pass. Each probability model requires 7 bits of storage. There are 13 models for the significance propagation pass, 3 for the magnitude refinement pass and 15 for the cleanup pass. Assuming that we have  $N$  magnitude bitplanes ( $N + 1$  bit signed subband data), there are  $N$  clean up passes,  $N - 1$  magnitude refinement passes and  $N - 1$  significance propagation coding passes per code-block. This gives a total of  $3N - 2$  passes per code-block.

Per code-block, we can write the amount of storage required to implement the EPSAE as

$$\begin{aligned} S_{\text{State}}^{\text{Block}} &= \frac{1}{8} [(N - 1) (13 \times 7 + 3 \times 7) \\ &\quad + N \times 15 \times 7 + 51 (3N - 2)] \\ &= 46.25N - 26.75 \text{ bytes} \end{aligned}$$

If  $W$  is the width of the image and  $K$  is the width of the code-blocks, there will be  $\lceil \frac{W}{K2^d} \rceil$  code-blocks along a line of subband samples in decomposition level  $d$ . Switching occurs between code-blocks in each subband, at each level of decomposition. We must therefore provide enough state memory for a row of code-blocks in each level of decomposition. The amount of pass state memory required is then

$$S_{\text{State}}^{\text{Total}} = 3 \sum_{d=1}^D \left\lceil \frac{W}{K2^d} \right\rceil (46.25N - 26.75) \text{ bytes}$$

For example, a 1024 sample wide image with  $64 \times 64$  code-blocks, 16 bit subband data and 5 levels of decomposition will require 31.27kB of state memory.

The PSAE of [2] provided three sets of the MQ registers and probability models, to allow for the parallel coding of each pass in the bitplane. If the EPSAE is implemented in

the same way, each of the three states must be read at the start of a group of stripes, and written at the end of each group, in each bitplane, with the exception of the most significant bitplane which has only a single pass. Note that no read is required when processing the first group of stripes in a code-block, and no write is required when processing the last group. The required memory bandwidth of the state memory is then

$$BW_{\text{State}} = \frac{2(\lceil \frac{J}{4S} \rceil - 1)(46.25N - 26.75)}{JK} \text{ bytes/sample}$$

For example, using  $64 \times 64$  code-blocks, 16 bit subband data and  $S = 1$  stripe, will require 4.89 bytes/sample.

Note that if less than three sets of MQ registers and probability models were provided, the bandwidth of the state memory would increase considerably. This is because we are processing the passes within a bitplane in parallel. Thus, each time we encounter a bit from a different pass, we would need to save the current state and load the new state.

The generation of context relies not only on the samples in the current stripe, but also those from the last line of the previous stripe. If we are processing the first stripe in a group, we must also have access to the line that preceded this group. We need to store the last line for each subband in each level of decomposition. Assuming 16 bit subband data, the size of the last line store will be

$$S_{\text{Last Line}} = 6W \sum_{d=1}^D 2^{-d} \lesssim 6W \text{ bytes}$$

For our example 1024 sample wide image, with 5 levels of decomposition, the last line store would require 5.81kB.

This line must be written at the end of each group of stripes (except the last stripe in a block), and read at the start of each group of stripes (except in the first stripe in each block). The bandwidth will thus be

$$BW_{\text{Last Line}} = \frac{2}{J} \left( \left\lceil \frac{J}{4S} \right\rceil - 1 \right) \text{ bytes/sample}$$

For our example 1024 sample wide image, with 5 levels of decomposition,  $64 \times 64$  sample code-blocks and using one stripe, the last line store would require a bandwidth of 0.47 bytes/sample.

## 6. COMPRESSED DATA MEMORY MANAGEMENT

When calculating the memory size and bandwidth requirements of the coder, previous work seems to have considered the buffering of coded data as external to the system, and as such not included it in the memory requirements. It is assumed that some external agent will take care of

buffering and arranging the compressed data into a compliant bitstream. However, for a fair comparison between the proposed coder and the loose and tightly coupled systems, we outline a simple management scheme and the additional costs involved for the agent.

We assume that a spacially progressive bitstream is produced. This type of progression means all quality layers are placed into the bitstream at once, so the agent can flush complete code-blocks out of the buffer. This progression may require in the order of thousands of lines of image data to be compressed before a precinct can be flushed, as code-blocks from each level of decomposition must be available. Due to this large number of image lines, there will be a considerable amount of compressed data to be buffered. This large amount of data will require the buffer to reside in external memory, regardless of the system under consideration.

The proposed system produces highly fragmented code-words. A fragment is produced for each pass, in each stripe, in each code-block, in each layer of decomposition. For these to be flushed into the bit stream, the fragments must be concatenated back into a single codeword. Each fragment is placed into the compressed data buffer, and its length (found in  $L$  register of the arithmetic coder) stored in a table. Individual tables for each level of decomposition are provided, to ease to location of required code-blocks. The tables are subdivided into stripes, and each stripe has pointer to its first fragment in memory. Assuming that the length of a fragment can be expressed using 8 bits, and that the pointer requires 32 bits, the total number of bytes required for a single code-block at level  $d$  of decomposition is

$$S_{\text{table entry, } d} = (3N - 2) \left\lceil \frac{J}{4S} \right\rceil + \left\lceil \frac{4}{\frac{W}{K2^d}} \right\rceil \text{ bytes/code-block}$$

For our example system at the first level of decomposition,  $64 \times 64$  sample code-blocks, single stripe processing and 16 subband data, each code-block stored will require 688.5 bytes. The total fragment table will be comparable to that of the compressed data, for the style of progression outlined.

The fragment entries will only be read and written once. The bandwidth of the table is then

$$BW_{\text{table, } d} = 2 \frac{S_{\text{table entry, } d}}{JK} \text{ bytes/sample}$$

Using the same example, this gives a bandwidth of 0.34 bytes/sample.

## 7. FINAL COMPARISON AND DISCUSSION

In this paper we have proposed a JPEG2000 encoding system which is able to operate with significantly less memory than systems which have been considered previously, while still maintaining loose synchronization between the DWT sub-system and a pool of block coders. Like the Pass

System	Memory Size	Memory BW
Loose	789 kB	8.79 bytes/sample
Tight	405 kB	3.42 bytes/sample
Proposed	106 kB	14.15 bytes/sample

Table 1: Comparison of memory requirements

Switching Arithmetic Encoder, the proposed system relies on the RESTART, RESET and CAUSAL mode switches, but exploits these both for parallel processing of the various coding passes and for partial processing of code-blocks in stripes.

Using the examples provided, the results in Table 1 are achieved. The proposed system substantially reduces the required memory, at the cost of a higher memory bandwidth. With memory size reduced, storage may now be implemented “on-chip”, making the increase in bandwidth of less consequence. The memory requirements of the each system are sensitive to code-block size. Memory size and bandwidth are favored by wider and shorter code-blocks.

The proposed system substantially increases the amount of memory required for compressed data storage. However, as this is likely to be ‘off-chip’ external memory, bandwidth is of a higher concern and this increase is quite small. The table system proposed is very simple and further study may yield a more efficient structure.

Although the proposed system maintains loose synchronization between the DWT and block coding sub-systems, we have not studied the impact of statistical fluctuations in the block processing time and DWT arrival pattern on the amount of additional buffering which may be required to maintain a given level of block coder utilization. Such an analysis should serve to further quantify the merits of the loosely coupled and proposed systems.

## 8. REFERENCES

- [1] D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, I. Ueno, and F. Ono, “Embedded block coding in JPEG2000,” in *Proc. IEEE International Conference on Image Processing*, vol. 2, pp. 33–36, 2000.
- [2] J. Chiang, Y. Lin, and C. Hsieh, “Efficient pass-parallel architecture for EBCOT in JPEG2000,” in *Proc. IEEE International Symposium on Circuit and Systems (ISCAS’02)*, vol. 2, pp. 773–776, 2002.
- [3] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practise*. Kluwer Academic Publishers, 2002.
- [4] K. Chen, C. Lian, H. Chen, and L. Chen, “Analysis and architecture design of EBCOT for JPEG-2000,” in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS’01)*, vol. 2, pp. 765–768, 2001.